Webinar

Recording in ON24

Pre-Recorded Events

Jonny Poole

Qlik Best Practices

Performance and Optimization

Jonny Poole

21 January, 2016

Jonny Poole

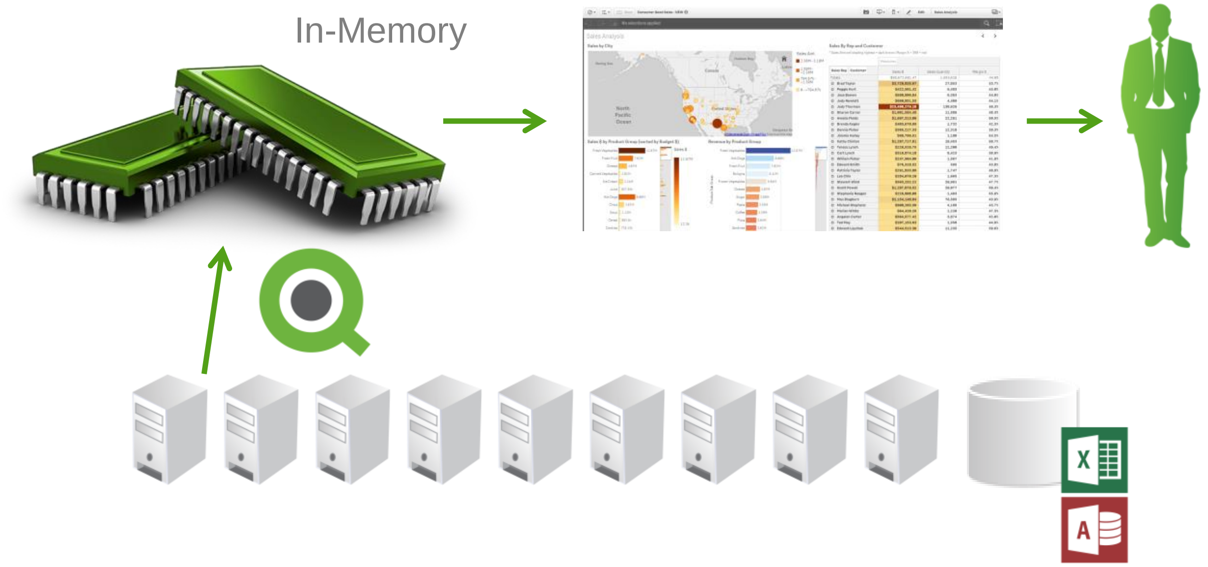
@Qlik since 2011
Principal Solution Architect
https://community.qlik.com/people/jpe

Agenda

- Section I Qlik's Core Technology
  - Best Practices for data modeling, expressions, data loading & app maintenance
- Section II Data in Motion
  - The art of platform selection and configuration
- Section III Data Segmentation
  - More data and how to reset the Qlik scalability
- Section IV Direct Discovery
  - An option for some scenarios
- Section V On-Demand App Generation
  - An elastic approach to analytics with massive data sets

Section I Qlik In-Memory approach

- Loads compressed data into memory
- Uses an *accelerated* compression algorithm
- Enables associative search and analysis
- Supports 100’s millions to billions of rows of data



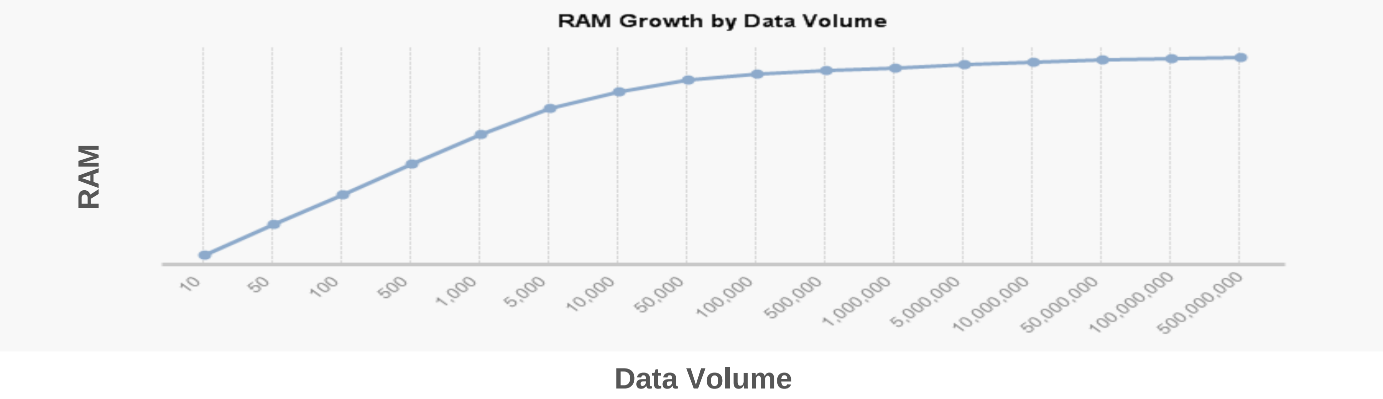
Qlik’s Secret Sauce

- To understand the **secret sauce** it’s important to understand how Qlik handles data.

- Qlik’s stores each unique data value once.
    - Product Names, Customer Names, Dates , Revenue amounts etc…

- Once the variety of unique values are understood, the need for more RAM drops dramatically

- This results in a very attractive scalability curve. Surprising ? How is that possible ? = **secret sauce**



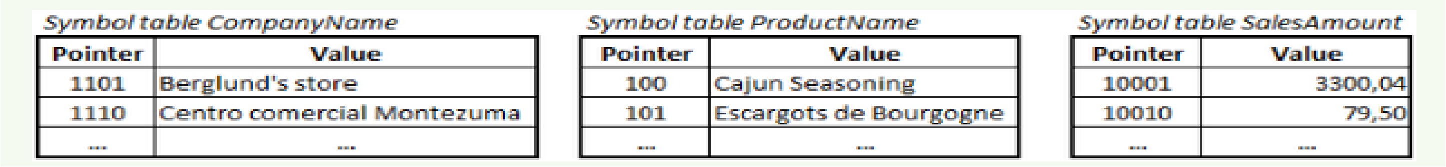
The Storage Model

Symbol Tables

- For each unique field in an associative database, Qlik creates a 2 column table call a Symbol table. The symbol table contains 1 record for each unique value loaded through the ETL process. One column stores the unique value and the other column stores a ‘symbol’ value. The symbol is a ‘bit stuffed pointer’

reference: Symbol Tables and Bit Stuffed Pointers

  – The storage need for a bit stuffed pointer value increases the bigger the pointer value (a high row number bit stuffer pointer value takes up more room)… so two reasons to reduce variety of values per field



Symbol table CompanyName

Pointer	Value
1101	Berglund's store
1110	Centro comercial Montezuma
...	...

Symbol table ProductName

Pointer	Value
100	Cajun Seasoning
101	Escargots de Bourgogne
...	...

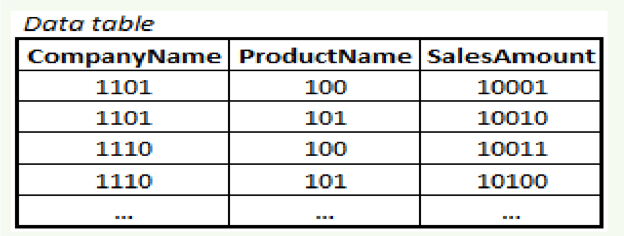
Symbol table SalesAmount

Pointer	Value
10001	3300,04
10010	79,50
...	...

and Data Tables

- While creating the symbol tables Qlik also recreates the actual tables, using the symbol values in lieu

- Remember, large bit stuffed pointer values will be repeated in the Data table… so three reasons to reduce variety of values per field ☺

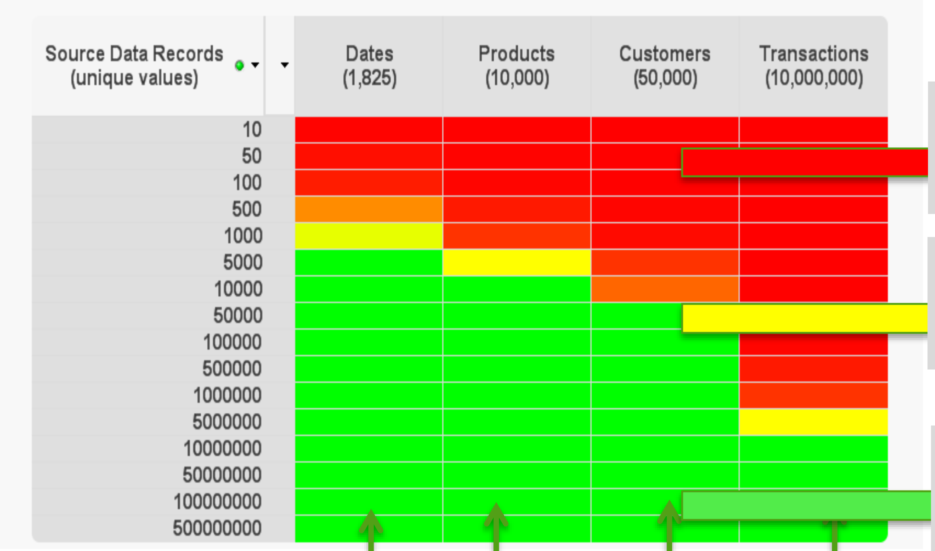


Data table

CompanyName	ProductName	SalesAmount
1101	100	10001
1101	101	10010
1110	100	10011
1110	101	10100
...	...	...

The Effect on RAM of adding more data

- At greater volume, the majority of unique value tables are saturated.
- As each Symbol table saturates, the demand for more RAM reaches inflection points and requirements taper…
- Saturation of Dimensional Data tables also occurs, and minimal growth is experienced



Source Data Records (unique values)	Dates (1,825)	Products (10,000)	Customers (50,000)	Transactions (10,000,000)
10	🟥	🟥	🟥	🟥
50	🟥	🟥	🟥	🟥
100	🟥	🟥	🟥	🟥
500	🟧	🟥	🟥	🟥
1000	🟨	🟥	🟥	🟥
5000	🟩	🟨	🟥	🟥
10000	🟩	🟩	🟧	🟥
50000	🟩	🟩	🟩	🟥
100000	🟩	🟩	🟩	🟥
500000	🟩	🟩	🟩	🟥
1000000	🟩	🟩	🟩	🟥
5000000	🟩	🟩	🟩	🟨
10000000	🟩	🟩	🟩	🟩
50000000	🟩	🟩	🟩	🟩
100000000	🟩	🟩	🟩	🟩
500000000	🟩	🟩	🟩	🟩

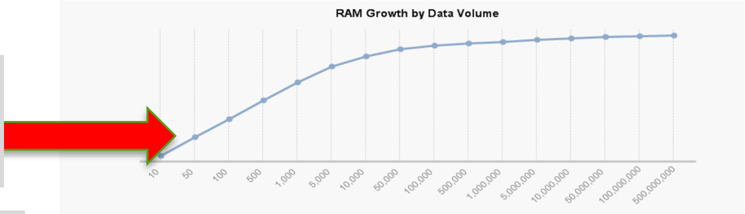
5 years = 1825 records

10,000 unique product names

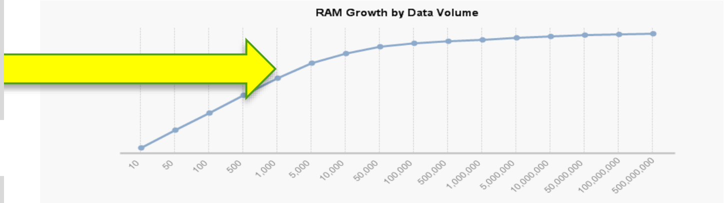
50,000 unique Customer names

10,000,000 unique transactions

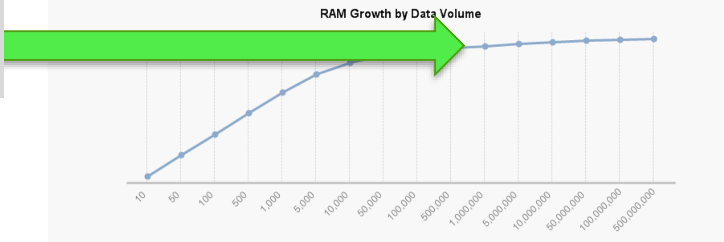
High RAM Growth



Tapering Ram Growth



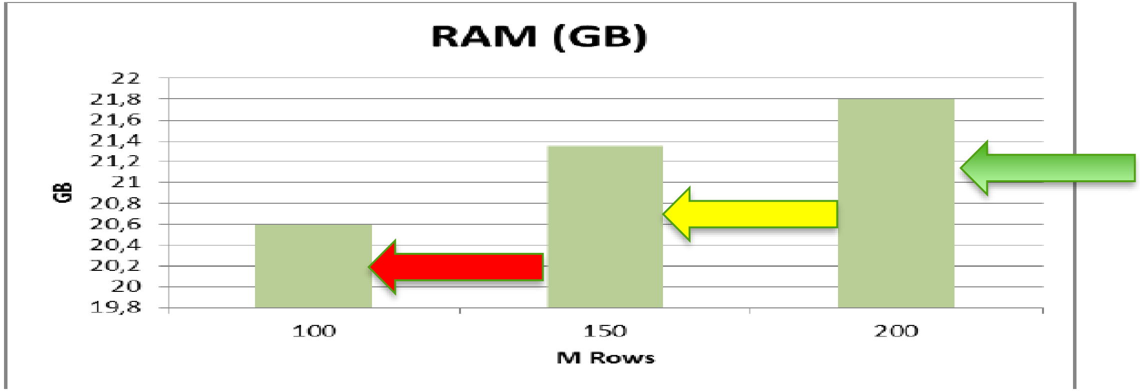
Minimal RAM Growth



Source: Qlik Sense Scalability Data Sheet

- As data volumes grow from from 100M to 200M records, only a fraction more RAM is required (ex: 20.6GB to 21.8GB).

- Source: Qlik Sense Scalability Data Sheet



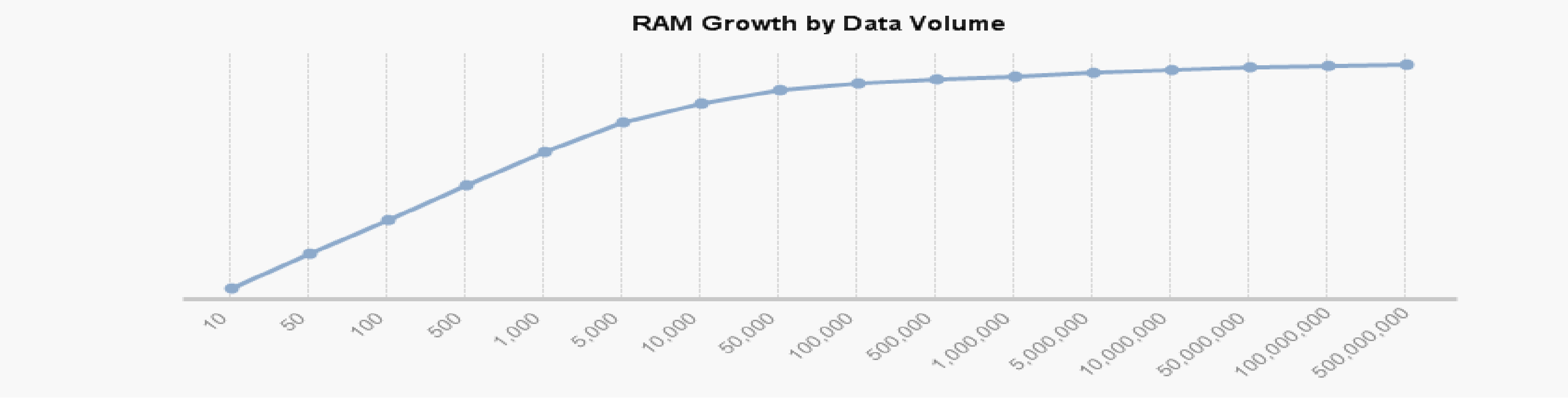
- 1.5x data = 3.9% RAM footprint increase

- 2x data = 5.8% RAM footprint increase

So what is the **Secret Sauce ?**

- -> A storage model that results in an accelerating compression model
- -> A scalability curve that levels down with more data

outstanding performance + reliable scalability = a **hit product**



Data Modeling best practices

Data Model performance

- Synthetic keys removed from data model
- Remove system keys/timestamps from data model
- Unused fields removed from data model
- Remove unneeded snow flaked tables (consolidate)
- Break concatenated dim. fields into distinct fields
- Remove link tables from very large data models, table concatenation is a possible alternative
- Use integers to join tables where possible
- Use ***Autonumber()*** to replace large concatenated keys

Best practices for **Data Model Performance**

- **Drop unused fields**
  - Avoid select *
  - Unused Fields app: http://qlikviewcookbook.com/recipes/download-info/document-analyzer/
    - Fewer fields = Data Tables that are less wide
    - Eliminates some symbol tables
- **Aggregate unnecessary detail**
  - Load dates instead of datetimes -> Date(Floor(Timestamp))
  - Changes ‘2015-01-01 12:00:00.000 TT’ to 2015-01-01’
  - 5 years = **~157,000,000** unique minutes vs. **1825** unique dates
    - Symbol table has far fewer records
    - Removes many larger bit stuffed pointer values from both symbol and data table
- **Replace ID fields and potentially any field that you need in the data model that won’t be displayed in the UI with integers by using autonumber() functions.**
  - The actual data value isn’t loaded, but a short integer is.
  - Are you displaying field values or just counting them ?
    - Symbol table will have a smaller footprint

Best Practices for **Expressions**

- **Minimize nested Ifs**

```
If( Region='North America',  100,
        if( Region='Asia',50,
                if( Region='Europe',75)
        )
)
```

- **Solution #1 -** pick(match()) can be a replacement in the UI

```
=pick( match( Region,'North America','Asia','Europe'),100,50,75)
```

- **Solution #2 -** Mapping load may also work in the load script

```
RegionMap:
Mapping Load * inline [
        Region,Number
        North America,100
        Asia,50
        Europe,75];

Regions:
        Load
          Region,
         applymap('RegionMap',Region) as Number;
        From <>;
```

Best Practices for **Expressions**

Minimize string comparisons

- **For example in Set Analysis or IF statements**

Worst:	Sum(If(ClientGroup=‘North America Sales’, Sales))
Better:	Sum(If (ClientGroupNum=1,Sales))
Best:	Sum({<ClientGroupNum={1}>} Sales)

```
Groups:
Load * inline [
ClientGroup
North America Sales
Europe Sales
Asia Sales];
```

```
Left Join (Groups)
Load * inline [
          ClientGroup, ClientGroupNum
          North America Sales,1
          Europe Sales,2
          Asia Sales,3];
```

Best Practices for **Expressions**

Works: Sum({<Date={“>=$(=Yearstart(max(Date)))<=$(=max(Date))”}>} Sales)

- **Use Boolean Flags**

Back End:

```
Orders:
Load
  Date;
From <>;
```

```
Orders
Load
  Date,
  if(Year(Date)=Year(Today()),1,0) as YTDFlag;
From <>
```

```
Date,YTDFlag
12/31/2015,1
12/31/2014,0
6/30/2015,1
```

Symbol table of Boolean flag is only 2 records

Common with relative time situations: YTD, Prior YTD, MTD, QTD, Last 6 months rolling etc…

Boolean Flags can eliminate unnecessary IF statements and Set Analysis

They are also have a light footprint in the data model making them an easy choice

Performs better with more Data: Sum({<**YTDFlag**={1}>} Sales)

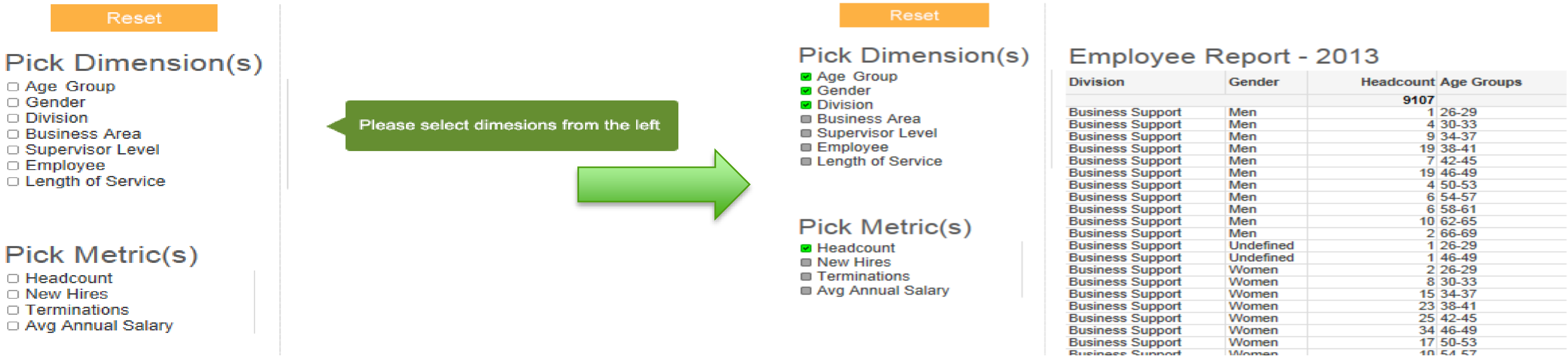
Performs best: Sum(Sales * **YTDFlag**)

Avoid/Manage large **straight tables** and pivot tables

- Users always want this so they can export all the data and work offline.
- But they are very expensive from a performance standpoint

Manage the request with the following design and strive to understand the business case

- Introduce UI design that allows selections for fields and measures
- Limit selections and records (via show/hide condition) as necessary
- Add Calculation Conditions to limit displayed rows if necessary



Pick Dimension(s)

□ Age Group
□ Gender
□ Division
□ Business Area
□ Supervisor Level
□ Employee
□ Length of Service

Pick Metric(s)

□ Headcount
□ New Hires
□ Terminations
□ Avg Annual Salary

Please select dimesions from the left

Pick Dimension(s)

☑ Age Group
☑ Gender
☑ Division
□ Business Area
□ Supervisor Level
□ Employee
□ Length of Service

Pick Metric(s)

☑ Headcount
□ New Hires
□ Terminations
□ Avg Annual Salary

Employee Report - 2013

Division	Gender	Headcount	Age Groups
		9107	
Business Support	Men	1	26-29
Business Support	Men	4	30-33
Business Support	Men	9	34-37
Business Support	Men	19	38-41
Business Support	Men	7	42-45
Business Support	Men	19	46-49
Business Support	Men	4	50-53
Business Support	Men	6	54-57
Business Support	Men	6	58-61
Business Support	Men	10	62-65
Business Support	Men	2	66-69
Business Support	Undefined	1	26-29
Business Support	Undefined	1	46-49
Business Support	Women	2	26-29
Business Support	Women	8	30-33
Business Support	Women	15	34-37
Business Support	Women	23	38-41
Business Support	Women	25	42-45
Business Support	Women	34	46-49
Business Support	Women	17	50-53
Business Support	Women	10	54-57

Best Practices for **Expressions**

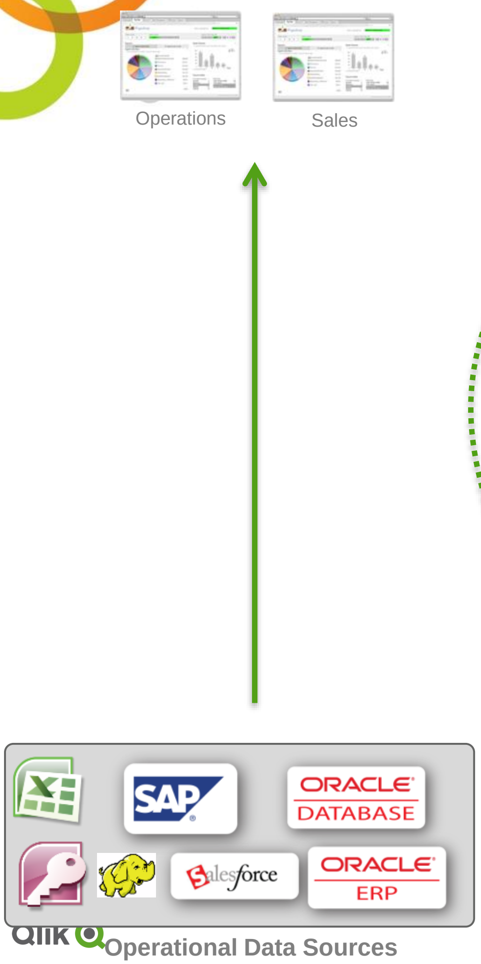
- **Avoid AGGR function when possible**
  - Can be difficult, sometimes you can load an aggregated table in the data model instead

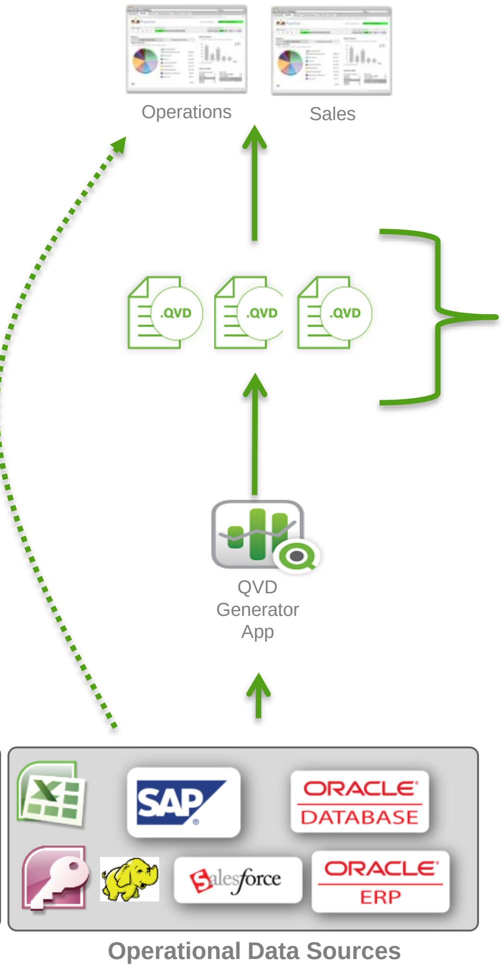
```
Orders:
Load
        OrderDate,
        Month(OrderDate) as OrderMonth,
        Sales;
From <>;
```

```
MonthOrders:
Load distinct
        OrderMonth,
        sum(Sales) as MonthlySales;
resident Orders
group by OrderMonth;
```

- **Avoid calculated chart dimensions if possible**
  - Try calculating in the load script if possible
- **Avoid/Manage Charts that use fields from 3-4+ different tables**
  - Includes situations where the fields come from tables that are several ‘hops’ apart in the data model

How about the data load ? QVDs

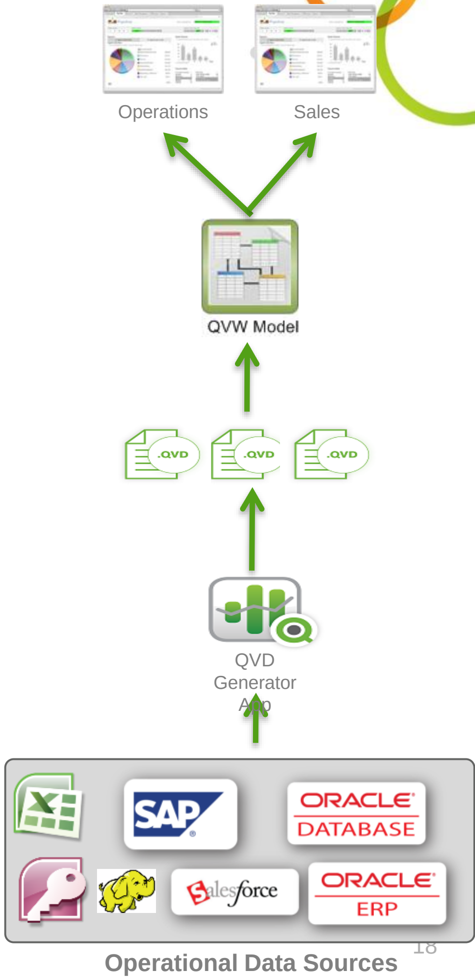




A QVD (QlikView Data) file is a file containing a table of data that QlikView has extracted from one or more data sources. QVD is a native QlikView format and can only be written to and read by QlikView. They are created with the scripting that is included in the QVW files.

Benefits:

- Single Source of Truth
- Incremental Loads
- Snapshots Loads
- Delivery Flexibility
- Very Fast Data Load



Loading Bottlenecks: QVDs

- Qlik loads are a serial process defined by a load script

- It is easy to create a load script that includes many data transformations and calculations that ends up taking a long time, especially when working with a large data set.

- QVDs allow us to use critical data loading optimization techniques:

  - Optimized QVD Loads
  - Incremental Loading
  - Parallel Loading

Loading Bottlenecks: QVDs

- Optimized QVD Loads
  - A QVD load is optimized when minimal transformations occur on the load
  - An optimized load will 100,000s of records per second
  - Use optimized loads to assemble a very large data.
  - Perform transformations in advance of QVD creation

- Incremental Load
  - Load Script has 3 steps:
    - Load history from QVDs (optimized load if possible)
    - Load and concatenate new/delta data set from source data sets
    - Rewrite the history to QVD (ready for next load)

- Parallel Loading
  - If you have a significant data volumes and/or significant transformations to process on the incremental load , the load into N loads

  Ex: Instead of loading 10 years of data in one load, load one year of data in 10 different loads and write to QVD

Loading Best Practices

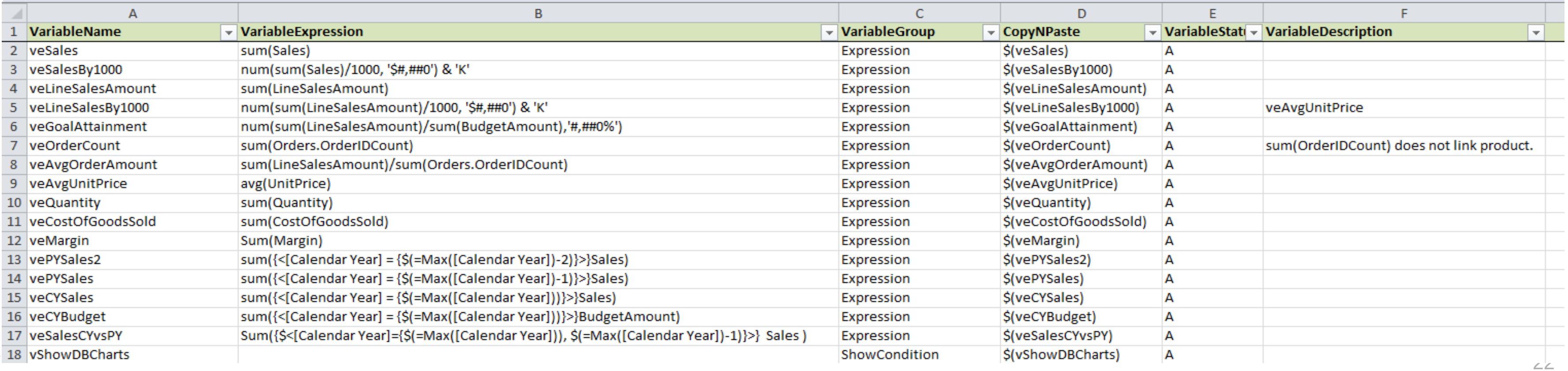
- Leverage QVDs
    - ‘Dynamic Data Update’ may offer a shortcut to incremental without QVDs

- Push transformations to database engines if possible

- Use different loading techniques to resolve different kinds of bottlenecks

Governance and Manageability

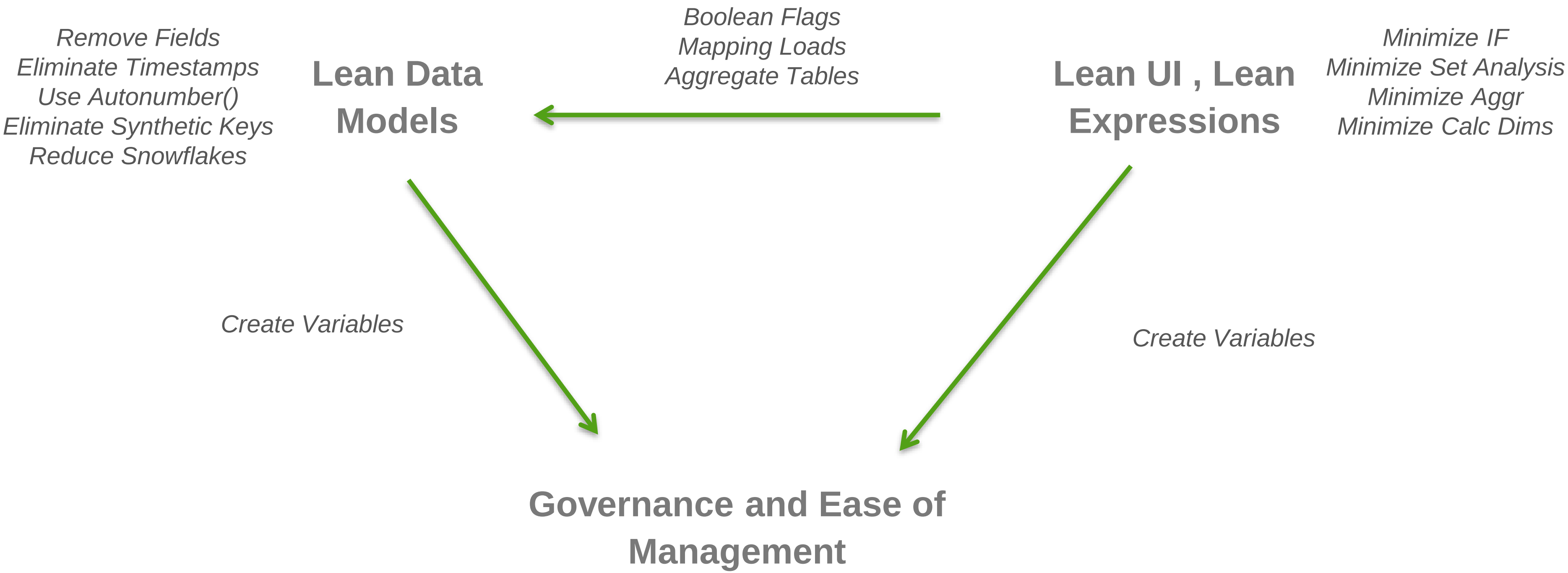
- Use Variables
- You can also externalize expressions used in a app , load them in a script using Let/Set commands as variables,
  - Allows the injection of common expressions into different apps to maximize reuse.
  - It also saves a developer from having to open the QVW in desktop and republish.



	A	B	C	D	E	F
1	VariableName	VariableExpression	VariableGroup	CopyNPaste	VariableStat	VariableDescription
2	veSales	sum(Sales)	Expression	$(veSales)	A	
3	veSalesBy1000	num(sum(Sales)/1000, '$#,##0') & 'K'	Expression	$(veSalesBy1000)	A	
4	veLineSalesAmount	sum(LineSalesAmount)	Expression	$(veLineSalesAmount)	A	
5	veLineSalesBy1000	num(sum(LineSalesAmount)/1000, '$#,##0') & 'K'	Expression	$(veLineSalesBy1000)	A	veAvgUnitPrice
6	veGoalAttainment	num(sum(LineSalesAmount)/sum(BudgetAmount),'#,##0%')	Expression	$(veGoalAttainment)	A	
7	veOrderCount	sum(Orders.OrderIDCount)	Expression	$(veOrderCount)	A	sum(OrderIDCount) does not link product.
8	veAvgOrderAmount	sum(LineSalesAmount)/sum(Orders.OrderIDCount)	Expression	$(veAvgOrderAmount)	A	
9	veAvgUnitPrice	avg(UnitPrice)	Expression	$(veAvgUnitPrice)	A	
10	veQuantity	sum(Quantity)	Expression	$(veQuantity)	A	
11	veCostOfGoodsSold	sum(CostOfGoodsSold)	Expression	$(veCostOfGoodsSold)	A	
12	veMargin	Sum(Margin)	Expression	$(veMargin)	A	
13	vePYSales2	sum({<[Calendar Year] = {$(=Max([Calendar Year])-2)}>}Sales)	Expression	$(vePYSales2)	A	
14	vePYSales	sum({<[Calendar Year] = {$(=Max([Calendar Year])-1)}>}Sales)	Expression	$(vePYSales)	A	
15	veCYSales	sum({<[Calendar Year] = {$(=Max([Calendar Year]))}>}Sales)	Expression	$(veCYSales)	A	
16	veCYBudget	sum({<[Calendar Year] = {$(=Max([Calendar Year]))}>}BudgetAmount)	Expression	$(veCYBudget)	A	
17	veSalesCYvsPY	Sum({$<[Calendar Year]={$(=Max([Calendar Year])), $(=Max([Calendar Year])-1)}>} Sales)	Expression	$(veSalesCYvsPY)	A	
18	vShowDBCharts		ShowCondition	$(vShowDBCharts)	A	

Qlik

Trade offs



Remove Fields
Eliminate Timestamps
Use Autonumber()
Eliminate Synthetic Keys
Reduce Snowflakes

Lean Data Models

Boolean Flags
Mapping Loads
Aggregate Tables

Lean UI , Lean Expressions

Minimize IF
Minimize Set Analysis
Minimize Aggr
Minimize Calc Dims

Create Variables

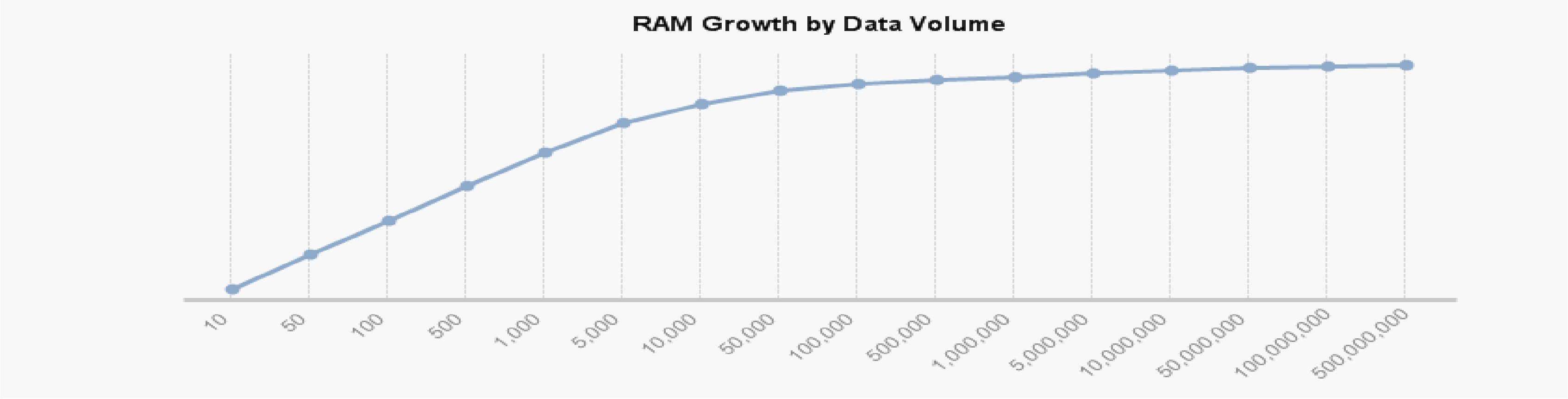
Create Variables

Governance and Ease of Management

Use QVDs

What have we learned so far?

- With a tuned data model and UI best practices, RAM is rarely a bottleneck for Qlik applications
- Next topic: so what is a bottleneck ??



Section II Data in Motion

- Qlik’s accelerating compression algorithm allows the QIX engine to store an enormous amount of data in memory. At the app level, RAM is rarely a bottleneck for Qlik

- Qlik’s engine is *sensitive* to certain chipset configurations that allow (or restrict) a large amount of in-memory data to move between RAM and CPUs

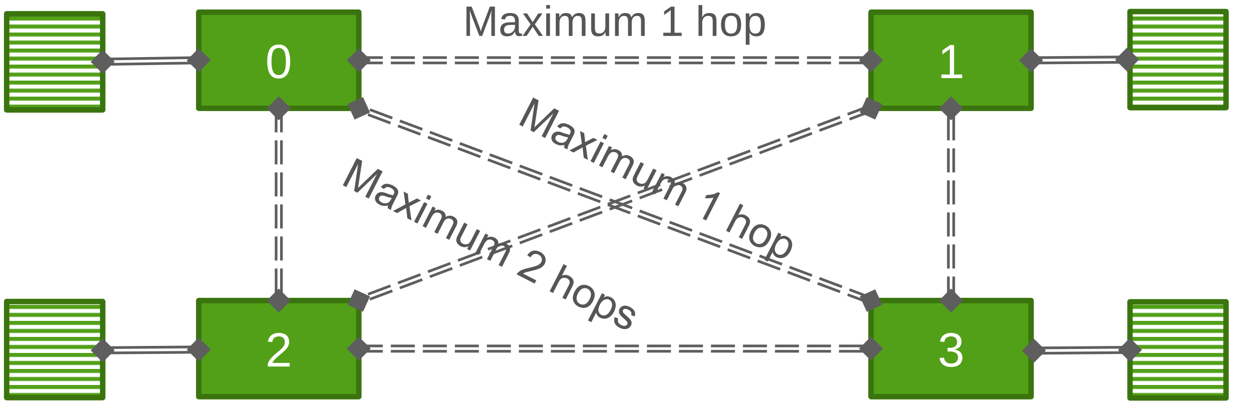
- With Qlik, CPU configuration is generally a *bottleneck* before RAM

- # QPIs are a key component of optimal CPU architecture and making the right hardware choice for qlik

https://en.wikipedia.org/wiki/Intel_QuickPath_Interconnect

The **Intel QuickPath Interconnect** (**QPI**)[1][2] is a point-to-point processor interconnect developed by Intel which replaced the front-side bus (FSB) in Xeon, Itanium, and certain desktop platforms starting in 2008.

Choosing hardware to maximize the number of QPIs



🟩 CPU

▤ RAM

◆═◆ Memory Bus

◆┅┅◆ QPI

Chipset	Example	Whitelisted
E5-26XX	DL380 G8	Yes
E5-46XX	DL560 G8	No
E7-48XX	DL580 G8	Yes

When you choose hardware for Qlik reduce ‘Data in Motion’ bottlenecks by…

- Selecting “white listed” hardware
  = Chipsets validated by Qlik Scalability Lab (see next slide)

- Avoid AMD

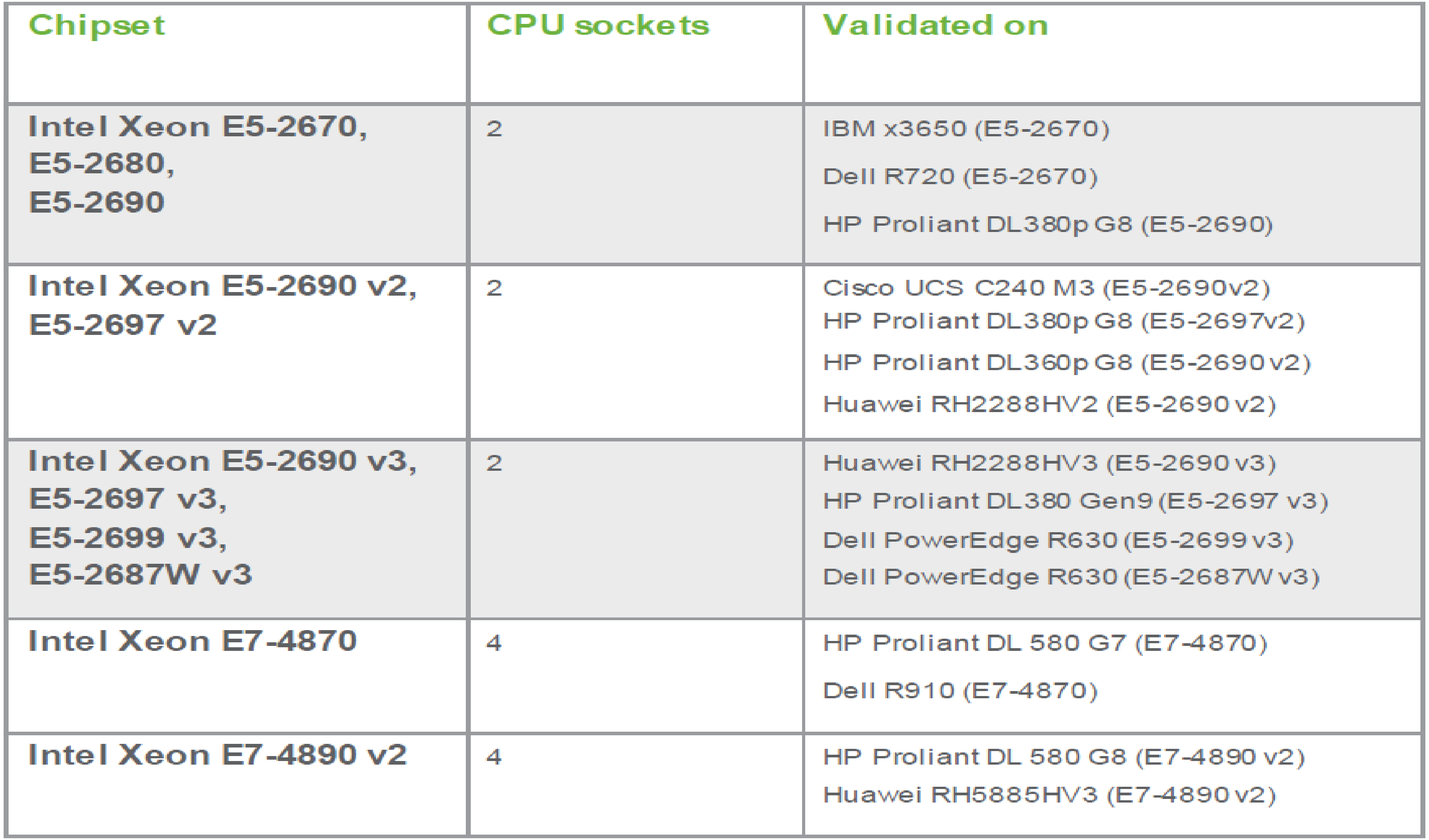
- With a white listed server, where possible, select the option with faster clock speed
  - Ex: E5-2680 or higher, including V2 chipset

- Install Memory according to manufacturer specs

- Avoid installing more memory such that bus speed drops. Memory bus speed is a priority
  - Ex: HP DL380 machines can run at 1866Mhz

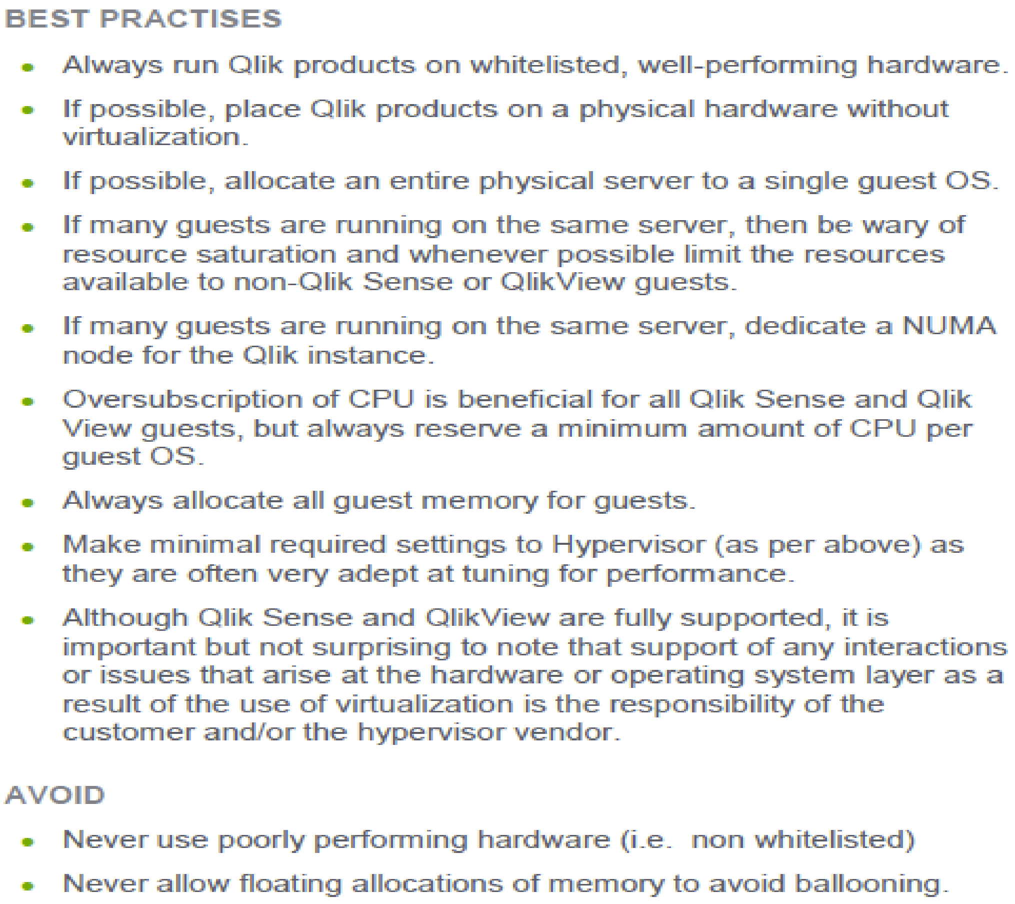
- Follow Qlik recommended server settings
  - Disable NUMA
  - Disable Hyperthreading except E5-26XX series
  - Power Mgmt = Max
  - Turbo Boost = Enabled
  - http://community.qlik.com/docs/DOC-2362

Chipsets that are well performing (updated June 2015)



Chipset	CPU sockets	Validated on
Intel Xeon E5-2670, E5-2680, E5-2690	2	IBM x3650 (E5-2670) Dell R720 (E5-2670) HP Proliant DL380p G8 (E5-2690)
Intel Xeon E5-2690 v2, E5-2697 v2	2	Cisco UCS C240 M3 (E5-2690v2) HP Proliant DL380p G8 (E5-2697v2) HP Proliant DL360p G8 (E5-2690 v2) Huawei RH2288HV2 (E5-2690 v2)
Intel Xeon E5-2690 v3, E5-2697 v3, E5-2699 v3, E5-2687W v3	2	Huawei RH2288HV3 (E5-2690 v3) HP Proliant DL380 Gen9 (E5-2697 v3) Dell PowerEdge R630 (E5-2699 v3) Dell PowerEdge R630 (E5-2687W v3)
Intel Xeon E7-4870	4	HP Proliant DL 580 G7 (E7-4870) Dell R910 (E7-4870)
Intel Xeon E7-4890 v2	4	HP Proliant DL 580 G8 (E7-4890 v2) Huawei RH5885HV3 (E7-4890 v2)

VM Best Practices



BEST PRACTISES

- Always run Qlik products on whitelisted, well-performing hardware.
- If possible, place Qlik products on a physical hardware without virtualization.
- If possible, allocate an entire physical server to a single guest OS.
- If many guests are running on the same server, then be wary of resource saturation and whenever possible limit the resources available to non-Qlik Sense or QlikView guests.
- If many guests are running on the same server, dedicate a NUMA node for the Qlik instance.
- Oversubscription of CPU is beneficial for all Qlik Sense and Qlik View guests, but always reserve a minimum amount of CPU per guest OS.
- Always allocate all guest memory for guests.
- Make minimal required settings to Hypervisor (as per above) as they are often very adept at tuning for performance.
- Although Qlik Sense and QlikView are fully supported, it is important but not surprising to note that support of any interactions or issues that arise at the hardware or operating system layer as a result of the use of virtualization is the responsibility of the customer and/or the hypervisor vendor.

AVOID

- Never use poorly performing hardware (i.e. non whitelisted)
- Never allow floating allocations of memory to avoid ballooning.

Section III Resetting the whole performance curve

- The aforementioned CPU bottlenecks occur on a *per app* basis.

- Qlik has the ability to pass context and filters from one app to a related app \
  ="Drill through Qlik style"
  - This is called 'Document Chaining'

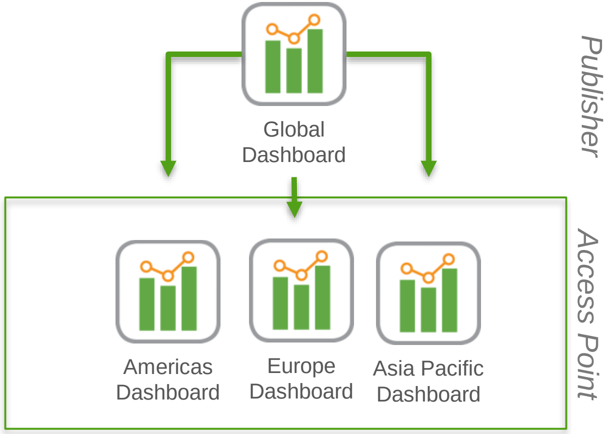
- QlikView also has the ability to auto-segment applications by a unique field identifier.
  - This segregates a large single document application into N documents
  - This is called 'Loop & Reduce'

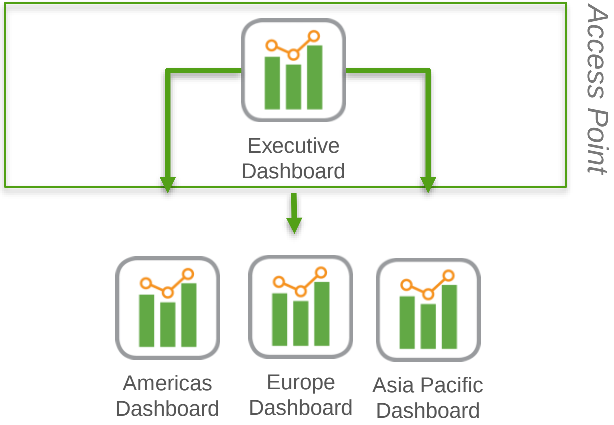
- Opportunity to reduce and even 'reset' symbol tables so that the applications leverage smaller bit stuffed pointer values to consume the same amount of data

Best practices for **Data Model Performance**

Very large QVWs represent a disproportionate amount of data (highly compressed), which will translate into a lot of data in motion. Even with optimal hardware, smaller QVWs have faster performance, so split one QVW into many:

- ***Loop & Reduce***
  - Allow Publisher to split large documents into many smaller components over a key value.
  - As a user looks at the dashboard, smaller documents are loaded in RAM





- ***Document Chaining***
  - Drill from one app to another passing selection state in the process
  - Keeps individual dashboards in smaller pieces

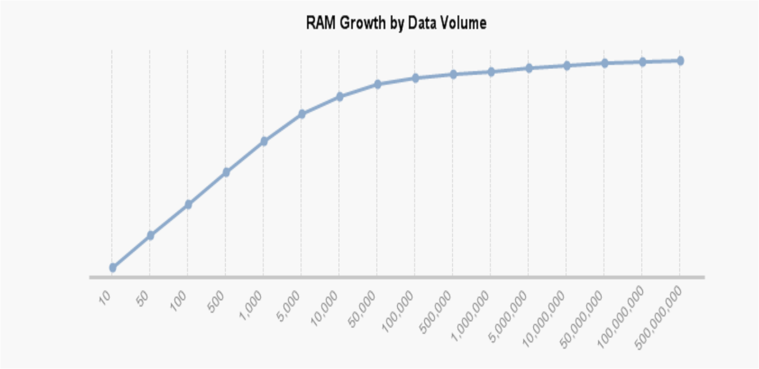
What does document chaining / loop & reduce do ?

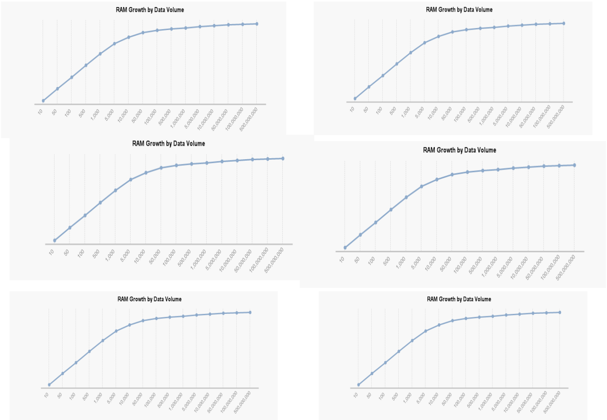
- You can push out CPU bottlenecking by a factor of ‘N’

  ‘N’ is the # of partitioned detailed applications produced by loop and reduce

  N is typically < 1000 …

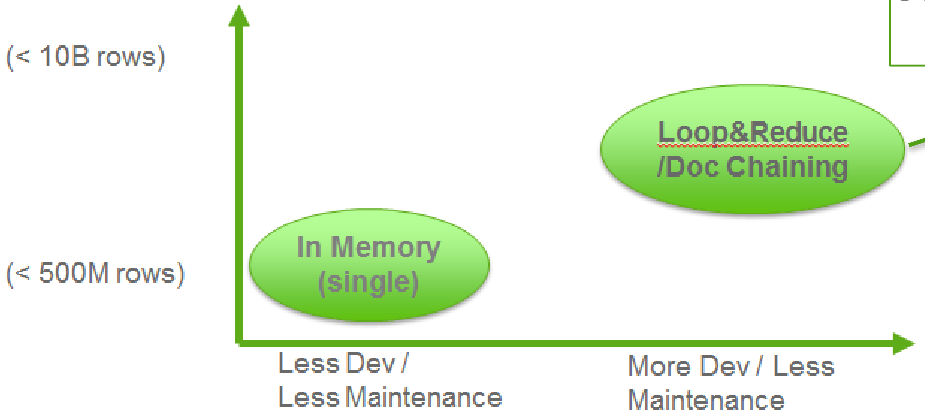
- <u>But</u>: you have to find a natural business line to ‘divide’ the analytics at the detail level





What can we do using exclusively in-memory applications?

- Conservatively: Several billion records
- Each application uses data modeling and UI best practices to reduce data in motion
- Use white listed hardware with optimal QPIs , memory bus speeds, and clock speeds
- We leverage in-memory where possible because the performance is amazing and recognized industry wide


(< 10B rows)
(< 500M rows)
Loop&Reduce /Doc Chaining
In Memory (single)
Less Dev / Less Maintenance
More Dev / Less Maintenance

Complex analytics at summary

- Analytics ‘partioned’ at detail

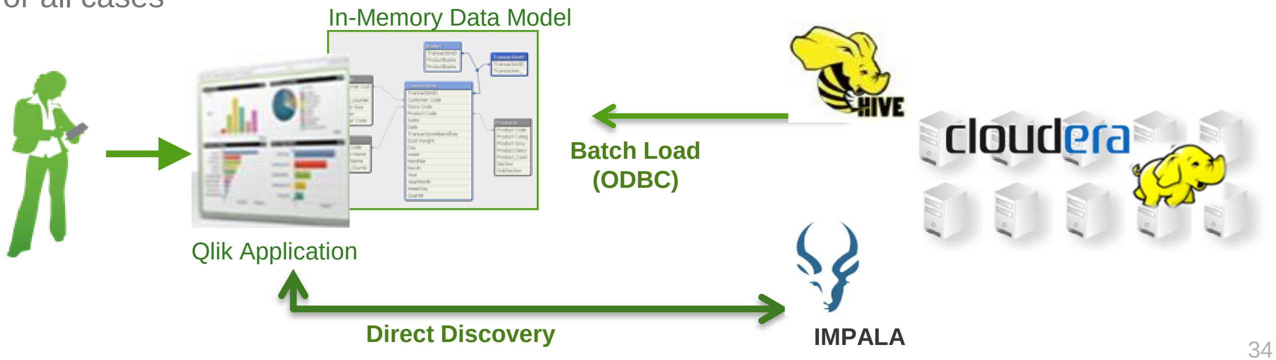
Qlik

Section IV Direct Discovery

- Direct Discovery is the ability to model a Qlik application WITHOUT pre-loading the data in memory
- It can help to make a larger amount of data available to a user. (It also helps with more near real time scenarios)
- Qlik auto-generates SQL that is sent to the database with every Qlik.
- Performance bottlenecking becomes largely controlled by SQL query performance against the backend database. (Not Qlik and no longer controlled by the Qlik Developer).

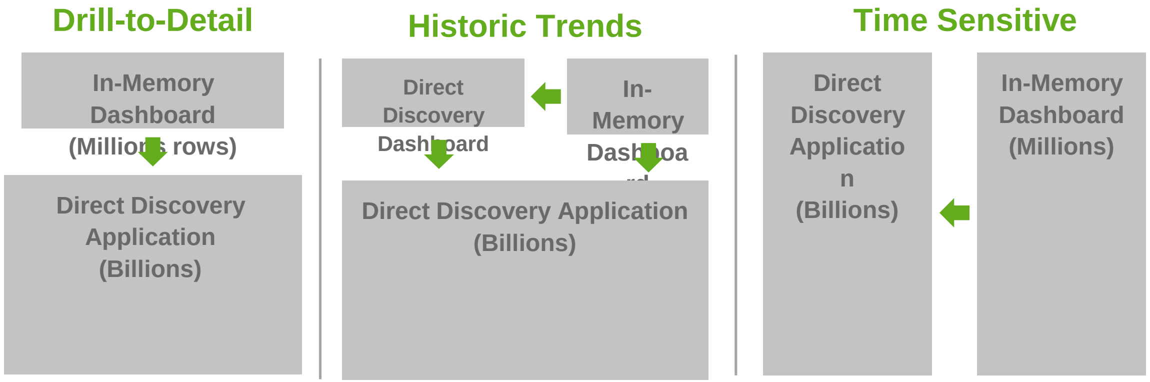
- But: direct discovery brings restrictions on expression/function capabilities due to the SQL dependency.

- So Direct Discovery is not a solution for all cases



Ways to leverage Direct Discovery

1. Build a single Qlik application using direct discovery
   - Limited expression library may limit analytics
2. Use with document chaining in lieu of ‘Loop&Reduce’
   - Summary applications have excellent analytics. Drill down to direct discovery app for ‘just the details’



Drill-to-Detail

In-Memory Dashboard (Millions rows)

Direct Discovery Application (Billions)

Historic Trends

Direct Discovery Dashboard

In-Memory Dashboard

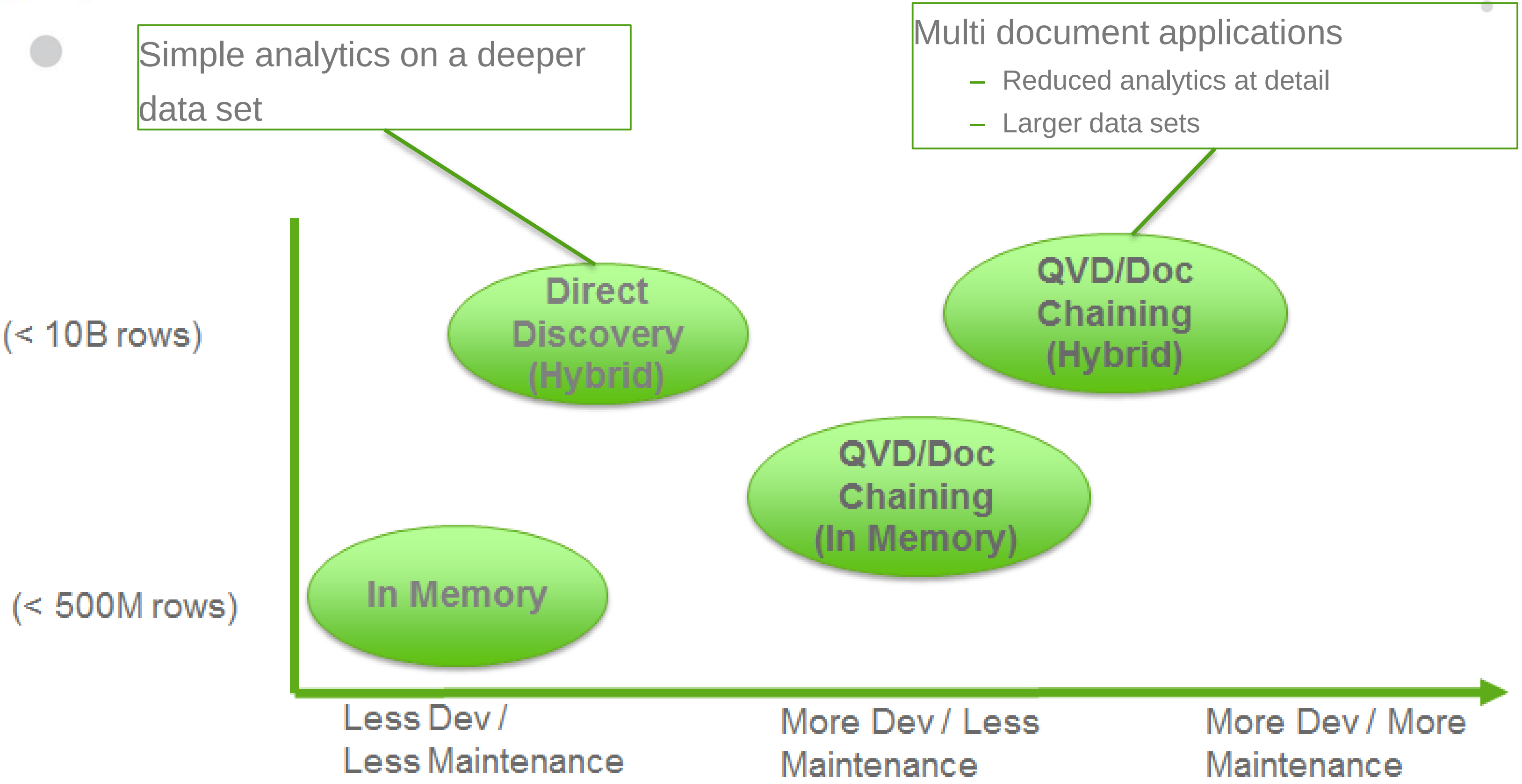
Direct Discovery Application (Billions)

Time Sensitive

Direct Discovery Application (Billions)

In-Memory Dashboard (Millions)

Effects of Direct Discovery



Simple analytics on a deeper data set

Multi document applications

- Reduced analytics at detail
- Larger data sets

(< 10B rows)

(< 500M rows)

Direct Discovery (Hybrid)

QVD/Doc Chaining (Hybrid)

QVD/Doc Chaining (In Memory)

In Memory

Less Dev / Less Maintenance

More Dev / Less Maintenance

More Dev / More Maintenance

There is another way to leverage these different approaches to achieve the most with Qlik with massive data volumes …

= On-Demand App generation

Section V On-Demand App generation

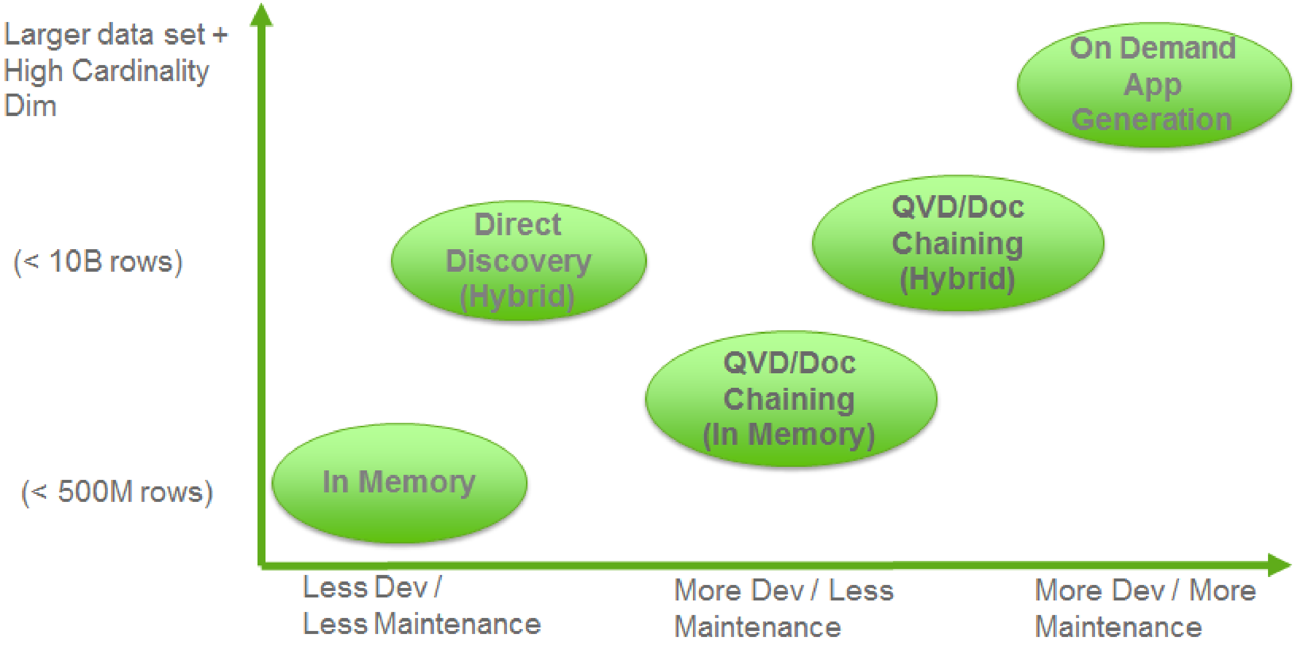
- Elastic Qlik analytics
- Use Direct Discovery to allow users to select fields and filters
- At the click of a button, a template Qlik application is loaded ‘on demand’ in-memory with the requested data set
- Provides users with detail record examination & dashboard visualizations
- Option to persist or throwaway
- Rapid spin up / spin down
- Shopping cart approach for users:

On-Demand App generation: what you get ?

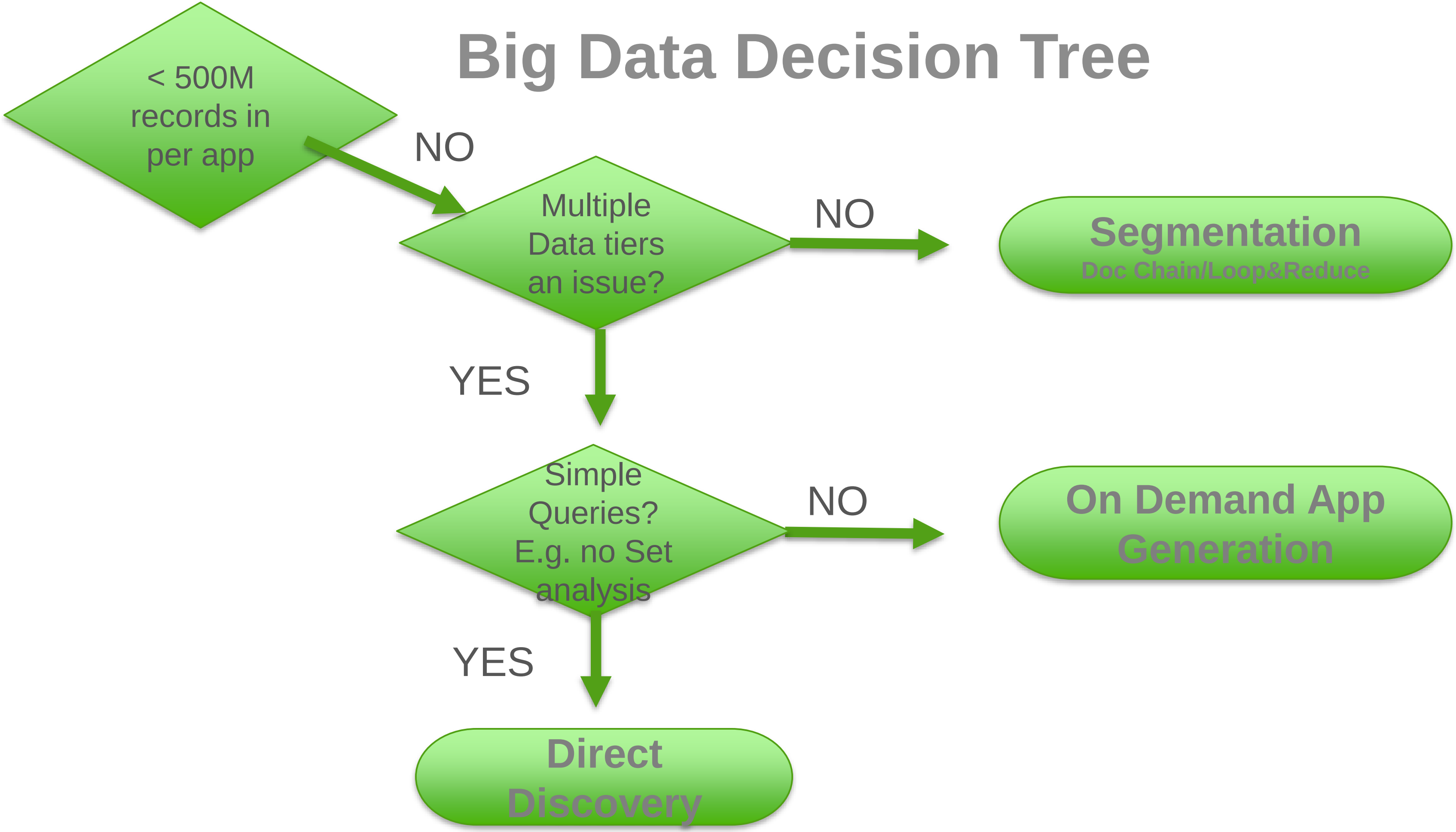
- **Data**:
  - Detailed records and billions of data points
- **Capability**:
  - Full breadth analytics and visualization
- **Performance**:
  - In-memory for the users
- **All the Best Practices:**
  - template uses data modeling best practices on whitelisted hardware

https://community.qlik.com/groups/financial-services-user-group/blog/2015/11/10/on-demand-app-generation?sr=stream&ru=2692

On Demand App Generation


Larger data set + High Cardinality Dim
(< 10B rows)
(< 500M rows)
On Demand App Generation
Direct Discovery (Hybrid)
QVD/Doc Chaining (Hybrid)
QVD/Doc Chaining (In Memory)
In Memory
Less Dev / Less Maintenance
More Dev / Less Maintenance
More Dev / More Maintenance

Big Data Decision Tree



< 500M records in per app

NO

Multiple Data tiers an issue?

NO

Segmentation

Doc Chain/Loop&Reduce

YES

Simple Queries? E.g. no Set analysis

NO

On Demand App Generation

YES

Direct Discovery

Qlik

Thank you